Software Architecture Description and UML

Paris Avgeriou[1], Nicolas Guelfi[1], and Nenad Medvidovic[2]

[1] Software Engineering Competence Center (SE2C), University of Luxembourg, 6, rue Richard Coudenhove-Kalergi, L-1359, Luxembourg
{paris.avgeriou, nicolas.guelfi}@uni.lu
[2] Computer Science Department, School of Engineering, University of Southern California, Los Angeles, CA 90089-0781 U.S.A
neno@usc.edu

Abstract. The description of software architectures has always been concerned with the definition of the appropriate languages for designing the various architectural artifacts. Over the past ten years, formal or less formal Architecture Description Languages (ADLs) and supporting methods and tools have been proposed by researchers. More recently, UML has been widely accepted in both industry and academia as a language for Architecture Description (AD), and there have been approaches to UML-based AD either by extending the language, or by mapping existing ADLs onto it. The upcoming UML 2.0 standard has also created great expectations about the potential of the language to capture software architectures, to allow for early analysis of systems under development and to support qualities. Furthermore, the latest trends such as MDA and the aspect-oriented paradigm are tightly connected with both UML and AD, thus promoting new approaches which combine the two. This workshop attempted to delve into this multi-faceted field, by presenting the latest research advances and by facilitating discussions between experts.

1 Introduction

Industry and academia have reached consensus that investing in architectural design in the early phases of the lifecycle is of paramount importance to the project's success [2, 4, 5, 7, 10]. Moreover an undoubted tendency to create an engineering discipline in the field of software architecture is apparent if we consider the published textbooks, the international conferences devoted to it, and recognition of architecting software systems as a professional practice [4]. Despite the attention drawn to this emerging discipline, there has been little guidance regarding how to describe a software architecture. Evidently there have been advances in the field, especially concerning design and evaluation methods, as well as reusable architectural artifacts such as architectural patterns and frameworks. And there is growing consensus nowadays about certain aspects of the task of software architecture description, such as the satisfaction of stakeholders' concerns through multiple views [1, 5]. But a software architecture needs to be rigorously described if we expect to benefit from its advantages such as communication of stakeholders, early analysis of the system, support of qualities and trouble-free maintenance. Unfortunately the problem of describing software architectures has not been solved; on the contrary we are still at early stages of addressing it [4].

One of the greatest challenges in describing software architectures, and a 'hot' topic of research nowadays, is the definition of the appropriate languages. The past ten years, formal or less formal Architecture Description Languages (ADLs) and supporting methods and tools have been proposed by researchers [8]. More recently, the Unified Modeling Language is being de facto accepted in both industry and academia as a language for Architecture Description (AD), and there have been approaches of UML-based AD either by extending the language, or by mapping existing ADLs onto it. The upcoming UML 2.0 standard has also created great expectations about the potential of the language to capture software architectures, to allow for early analysis of systems under development and to support qualities. Furthermore, MDA and the Aspect-Oriented paradigm are tightly connected with both UML and AD, thus promoting new approaches which combine the two.

The workshop on Software Architecture Description and UML made an effort to look into these issues from a holistic viewpoint inside the UML community. It has brought together researchers and practitioners who work on diverse aspects of Architectural Description (AD) of software systems, related to the Unified Modeling Language. It thus fostered a presentation of the latest approaches on the field from both industry and academia, as well as a creative discussion between the participants in specific themes.

The rest of this workshop report is organized as follows: Section 2 presents the theme of the keynote speech, which discussed the upcoming UML 2.0 standard with respect to specifying and enforcing software architectures. Section 3 outlines the contents of the papers that were presented in the workshop and involved the issues of components, connectors, architecture-based analysis, static and dynamic modeling of architectures with UML 2.0 and a development methodology that combines Aspect-Oriented Modeling and MDA. Section 4 describes the findings of the discussion panel that consisted of three invited experts, who discussed architectural issues from the viewpoint of three respective qualities: security, mobility and performance. Finally Section 5 concludes with a brief synopsis of the state-of-the-art and future trends.

2 Specifying and Enforcing Software Architectures

The keynote speech[1] concerned the theme of *specifying* and *enforcing* software architectures during the development and evolution of the system. Current software architecting practice often fails in both these activities: first, architecture is not explicitly specified, which results in architectural intent being 'hidden'

[1] The keynote speaker, Bran Selic, is an IBM Distinguished Engineer and the chair of the OMG task force, responsible for the finalization of the UML 2.0 standard.

or possibly 'buried' inside the code; second, as a result of this lack of architecture specification, the architecture cannot be enforced, i.e. it cannot be properly implemented and maintained. It thus runs the risk of getting corrupted by developers that don't understand it, even by minor changes such as bug fixes. This results in 'architectural decay', where the system implementation gradually drifts apart from the original architectural intent.

Architectures are meant to be modeled at different levels and different languages, including the code level in a programming language. Models can therefore be refined continuously at various levels of detail, from different viewpoints, until the system is fully specified at the code level. In this respect, a software system is distinguished from other engineering products, by the unique characteristic that the model per se *evolves* into the system implementation. The model-driven development paradigm implements this principle, based on two complementary techniques: *abstraction* that is supported by modeling languages, and *automation* of the transformations between the models, that is provided by tools. Thus, enforcing the architecture can be much more straightforward, since the architectural decisions can be passed on to the system through code generation. The benefits are increased productivity and assured quality, since it will then be impossible to corrupt the architectural intent by low-level programming.

The following definition of an *engineering model* was proposed: "a reduced representation of a system that highlights the properties of interest from a given viewpoint". This definition emphasizes the following aspects: that the model is an *abstraction* of the system at a specific level of detail; that it is often looked upon from different *viewpoints* that demonstrate different sides of the system; and that *representing* the system is not merely "syntactic sugar" but a meaningful visual aid. A software architecture in particular is a model that enables communication between the different stakeholders, drives the construction of the system and determines the system's capacity for evolution growth.

The rest of the discussion focused on the run-time view of software architectures, which deals with the run-time organization of significant software components interacting through interfaces, and being composed of successively smaller components and interfaces. The application of UML 2.0 in describing run-time architectures was elaborated. First, it was stressed that run-time architectures should not be modeled only statically through class diagrams but also at an instance level through collaborations. Then the most fundamental new concept in the upcoming UML 2.0 standard for architectural description was discussed: *structured classes*. These are originated from Architecture Description Languages and describe the inner structure of a class, either through a behavior specification or through a collaboration of parts through connectors. It is highly recommended that architects use structured classes to describe the hierarchical decomposition of systems' run-time structures. Ports are also a key concept in structured classes, since they are points of grouped interactions, they specify provided and required interfaces, and they decouple the structured class from external entities. Structured classes are joined by connectors through their ports, and connectors in turn can be constrained by a specific behavior protocol that can be appropriately specified with the use of interaction diagrams. The importance of structured classes lies in the fact that they can be rigorously specified and thus facilitate code generation in order to *enforce* the architecture. Finally, the Component element has been "upgraded" in UML 2.0 to subclass Structured Class and to allow for mappings to specific platforms (e.g. EJB).

As a concluding remark, it was stressed that "to architect is to model". The process of architecting is inherently a modeling activity which captures the architectural intent and subsequently enforces it during system development and evolution, thus preventing 'architectural decay'. Model-driven technologies are a promising approach in the software architecture field, and UML 2.0 in particular, encapsulates much of what was defined in classical architectural description languages and also supports architectural enforcement.

3 Issues in Software Architecture Description with UML

In order to facilitate the presentation of key topics in the field and to allow for extensive discussion on them, only six papers were selected to be presented to the workshop. The papers were chosen through a rigorous reviewing process, aimed at singling out high-quality submissions that concern a wide gamut of research issues: components, connectors, architectural analysis, architecture description in industrial projects, behavioral modeling and new trends such as Aspect-Orientation and MDA.

3.1 Documenting Architectural Connectors with UML 2

The paper by Ivers et al. discusses the issue of UML 2 support for Architectural Connectors, a concept which is treated by the software architecture community as a first-class entity, just like components. The authors recollect that UML 1.x was an awkward fit in representing architectural connectors, which led to designers making their own conventions, either by using the existing UML elements, or by extending the language. There was much anticipation in the architecture community to see whether the upcoming UML standard would provide a better support for connectors. The authors examine the concept of connectors in UML 2 with respect to how well it satisfies 4 criteria:

- *semantic match* - connectors naturally signify pathways of interaction.
- *visual clarity* - connectors should be distinguishable from components and be represented by a minimum number of visual elements.
- *completeness* - connectors should be able to represent behavior, state and interfaces.
- *tool support.*

The authors briefly analyze to what extent these criteria are fulfilled by four standard UML 2 elements, which could be used as connectors, namely Associations, Association Classes, Classes and Connectors. Their findings are that none of these elements is a perfect match, instead there is a tradeoff in using each one of them. The authors conclude that even though UML 2.0 is much more apt for architectural documentation in several aspects, representing connectors still seems to be problematic. It must be noted that the analysis presented in this paper focused on standard UML elements, and not on extensions of the language.

3.2 Using UML for SA-Based Modeling and Analysis

The paper by Cortellessa et al. reports on how their research group is using UML to specify Software Architectures (SA) for different kinds of analysis. They outline four different approaches related to SA-based model-checking, testing, performance and reliability analysis respectively:

– *Model checking* - It is performed through the *Charmy* framework that aims to assist the software architect in designing Software Architectures and in validating them against functional requirements. Formal model checking techniques are used to check the consistency between the SA models and functional requirements. The description of the architecture is based on stereotyped class diagrams for the component and connector view, state machines for the component behavior and scenarios for the specification of temporal properties.
– *Testing* - It aims to check to what extent a system under implementation conforms to its architectural specification. It offers the advantage of testing early and at a higher-level of abstraction. It allows the detection of structural and behavioral problems from UML stereotyped class diagrams and state diagrams respectively, as well as the specification of test cases as sequence diagrams. Test cases are firstly specified at an architectural level and then refined into the code level.
– *Performance analysis* - It is achieved through the SAPone approach which automatically generates a performance model, based on a Queueing Network model (QN), from a SA specification described by UML 2.0 Diagrams. The UML profile for Schedulability, Performance and Time (SPT) is utilized in order to annotate the UML diagrams with performance-related information.
– *Reliability analysis* - It focuses on modeling the reliability of a system as a function of the reliability of individual components and connectors. The authors have proposed an extension of UML to represent concepts in the reliability domain, especially for component-based systems, and thus produce *reliability models* at an architectural level.

Finally the authors introduce their ongoing work which aims to provide a framework for incorporating all the above approaches into the same analysis framework. Their rationale is based on the need to tradeoff between functional and non-functional properties, by integrating the analyses of individual properties. They have introduced a framework that aims at such an analysis integration, *independently* of the notations or languages used for the different kinds of properties.

3.3 Flexible Component Modeling with the ENT Meta-model

The paper by Brada identifies two problems in current component meta-models: (i) they merely reflect the present state-of-the-art in component technology without allowing for extensions that could accommodate future developments; (ii) the visual languages associated with the meta-models, similarly, offer specific, preset views on components rather than more adaptable visualizations. The author proposes an approach in order to alleviate both these deficiencies:

- by introducing the ENT component meta-model which is open to future technological developments and which enables us to define the component characteristics from the users point of view (rather than in just technological terms). This meta-model is built upon an analysis of a number of research and industrial component meta-models.
- by proposing a flexible graphical notation that, based on the meta-model abstractions, allows the users' to adjust the visual representation of component interfaces. This concept is similar to using multiple views for showing different aspects of a system's architecture.

The author advocates that this approach would allow present or future component metamodels to be mapped to the ENT metamodel, even if such mappings always entail semantic gaps. Finally, the combination of components specified in this metamodel with architectural connectors would be a challenging field of future research.

3.4 Designing the Software Architecture of an Embedded System with UML 2.0

The paper by Frick et al. discusses the results of an industrial project for model-driven development of embedded systems software. Part of this methodology was to devise an architecture description language, based on selected elements of UML 2.0, particularly leveraging the port concept. The authors focused on describing the software architecture of embedded systems as interconnections of modules through explicitly-specified *provided* and *required* interfaces. Pairs of required and provided interfaces are perceived as *contracts* that the module must conform to, and they are usually attributed to the module's ports. Thus a module imports or exports a service specified by a contract, through a port. Furthermore a module implementation can be either: (i) a behavioral model in terms of a state machine that implements the module services; (ii) a composite module that has an internal structure as mandated in the UML 2.0 composite structures package; (iii) code written in a programming language and wrapped in the context of UML. The first two cases support code generation, therefore, all three implementations are considered executable.

Another significant aspect of this approach is that it aims at product-family architecture design, where individual products, or *variants* are specific configurations of module variants. Subsequently the latter are different implementations of the same interface. This is a very helpful concept in the development of embedded systems, as environment components can be considered as variants and they can be simulated in order to test embedded control software.

3.5 Behaviors Generation from Product Lines Requirements

The paper by Ziadi et al. draws upon the current research trend to model variability in Product Lines (PL). Related research work has so far concentrated on the *static* architecture of PL; the authors extend it to the *behavioral* aspects. In specific the authors propose an approach to derive the behavioral specification of individual products from that of a Product Line. To begin with, they exploit the ability of UML 2.0 to algebraically compose sequence diagrams through special composition operators. Therefore, they specify PL behavioral requirements as algebraic expressions extended with constructs to specify variability. Building on that, they synthesize the detailed behavior for each product member in the PL in two stages: The first stage uses abstract interpretation of the variability operators in scenarios to get behavior specialization of the PL according to given decision criteria; in the second stage, the resulting product behavior specifications, expressed as sequence diagrams, are synthesized into statecharts.

This approach thus helps to refine behavioral specifications for the whole product family, which are specified in high-level sequence diagrams, into product-specific implementation-level statecharts. Therefore it fosters efficient, formalized traceability between requirements on a Product Line level and detailed design of individual products in PL. It can also promote reuse of statecharts between products that share common behavior.

3.6 A UML Aspect-Oriented Modeling Approach for Model-Driven Software Development

The paper by Vachon and Mostefaoui introduces a development methodology that combines Aspect-Oriented Modeling and Model-Driven Architecture (MDA), which have both received growing interest from the research community. The authors claim that these approaches naturally complement each other: MDA separates the business model, the computation model and platform specific-design decisions into distinct development steps and documents the transformation from one to another; aspect-orientation separates core functional requirements from “crosscutting application concerns” while at the same time merging them in a clean and explicit manner. Consequently, combining the two approaches entails the ‘weaving’ of aspects in the different MDA models and supporting the transformations among them.

Their method supports an iterative stepwise refinement process that not only takes care of the satisfaction of functional requirements in an MDA fashion, but also introduces aspects early: these are woven into platform-independent design decisions and then transformed to platform-specific models. From the aspect-orientation side, the authors propose a UML Profile as a modeling notation, called Aspect-UML, for the specification of aspects and their join points. From the MDA side they present the MDA-based development phases, focusing particularly on the transformation of platform independent models (PIM) into platform specific models (PSM). In specific they explain how to transform Aspect-UML models into selected PSM, using a mapping between their corresponding metamodels. They also explain how new generation transformation tools can potentially automate the transformation of an Aspect-UML PIM into target PSM.

4 Architectural Support for Qualities

The aim of the discussion panel was to discuss critical, but under-addressed issues pertaining to software architectural description. Three distinguished experts were invited to the discussion in order to shed some light on the architectural support for 3 respective qualities, namely security, mobility and performance. The short talks of the experts and the subsequent discussions are summarized in the following paragraphs.

Dr. Jan Jürjens[2] explored the field of architectural design for security-critical systems [6]. Dr. Jürjens advocated that the main problem in the software architecture of security-critical system is that security is not designed up-front as an architecture-level issue, but rather "circumvented" at a later stage, resulting in potential security compromises. The remedy that is proposed for this problem, is an approach, entitled *Model-based Security Engineering*. It deals with architectural design artifacts arising in industrial development of security-critical systems (e.g. UML models) and requires tool-supported security analysis. It mandates the automatic analysis of models against security requirements and then follows a round-trip engineering style, where code or tests are generated from models and vice versa. The approach suggests the use of UML for the typical reasons of standardization, broad industry adoption and extensive tool support. A UML profile, named UMLsec, has been proposed in order to grasp the details of secure systems development. Finally this approach suggests the use of *secure architectural patterns* in a formal, methodological way, using the aforementioned UMLsec profile.

Professor Raffaela Mirandola[3] elaborated on the issue of mobility of software systems. She advocated that mobility of code is an architectural-level design issue that serves several goals such as service customization, dynamic functionality extension, fault-tolerance, performance improvement etc. Unfortunately there is no silver bullet in designing architectures of mobile systems, instead there is always the risk of performance shortcomings. She also explained that the current architectural styles for mobile systems can be classified into two categories: those where only code moves and those where code moves along with its state. As far as the *locations* where mobility of software takes place, they can be logical or physical, they can be nested, and finally they can also be mobile themselves. There are currently two approaches to modeling architectures of mobile systems

[2] Dr. Jürjens is affiliated to the Technical University of Munich, Germany. email: juerjens@in.tum.de

[3] Professor Mirandola is affiliated to Universita di Roma "Tor Vergata", Italy. email: mirandola@info.uniroma2.it

[9]: (i) UML-based modeling which is visual, extensible, a de-facto standard in industry for architectural design but has imprecise semantics; (ii) "mobility-oriented" Process Algebras which are unambiguous, have compositional features, and facilitate analysis, but are overly complex, not widely used and they lack support for architectural design. Typically, as in other cases, bridging between formal and semi-formal approaches is a key research issue in this area.

Professor Murray Woodside[4] tackled the issue of performance modeling [11] with respect to software architecture modeling. He stressed the fact that the relation between the performance model and the architecture is bi-directional: the performance model is constructed upon the architectural model, and the results of performance modeling are a valuable feedback in selecting and validating the various design choices in the architectural model. Performance modeling is actually based on architectural high-level information such as architectural styles, partitioning of functionality into components etc., but it also requires additional low-level details, such as workload and demands for operations. Analysis of performance models subsequently takes place through formal techniques such as queueing, petri nets, layered queueing, simulation etc. An interesting aspect that arises from performance modeling is that different architectural configurations can be compared against each other, as long as some parameters such as workload, the platform and the number of processors are kept invariant. However, it is of paramount importance to evaluate the tradeoff between the detail and accuracy (and therefore cost) of performance modeling and the value of the produced results. A useful rule of thumb in this case is to match the precision of performance data to the level of detail in the architecture model.

5 Epilogue

We can safely conclude that the description of software architectures is still a very relevant subject in the research community. The practice of software architecting is growing, and there are many notations used in the scope of architectural description. UML is gaining more and more prominence and has made steps forward in this direction but can still be awkward to use for certain aspects of architectural description. The support for qualities has been under-represented in ADLs in the past, and this has not changed with UML; nor will the use of UML per se provide such support. A synergy between experts in the domains of the various qualities and software architecture, is a challenging issue and a necessity. The UML 2.0 standard is currently being explored for its appropriateness in the field, while some shortcomings have already been identified and attempts are made to overcome them. Nevertheless UML 2.0 is likely to redraw the landscape substantially. We are looking forward to this development, and will gauge UML 2.0's native architectural support, and software engineering community's reactions to it in deciding on possible follow-ons to this workshop.

[4] Professor Woodside is affiliated to Carleton University, Canada.
email: Murray.Woodside@sce.carleton.ca

References

1. Avgeriou, P., Guelfi, N., Razavi, R.: Patterns for documenting software architectures. Proceedings of the 9th European Pattern Languages of Programming (EuroPLOP) conference. July 2004, Irsee, Germany.
2. Bosch, J.: Design and Use of Software Architectures. Addison-Wesley, 2000.
3. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M.: Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. John Wiley and Sons, 1996.
4. Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J.: Documenting Software Architectures: Views and Beyond. Addison-Wesley, 2002.
5. IEEE, Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE std. 1471-2000, 2000.
6. Jürjens, J.: Secure Systems Development with UML. Springer-Verlag 2004.
7. Kruchten, P.: The 4+1 view model of architecture. IEEE Software, November 1995.
8. Medvidovic, N. and R. N. Taylor: A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Trans. Softw. Eng. 26 (2000), pp. 70-93.
9. Grassi, V., Mirandola, R., Sabetta, A.: A UML Profile to Model Mobile systems. In Proc. of UML 2004 conference, 11-15 October 2004, Lisbon, Portugal. Springer LNCS 3273.
10. Shaw, M., Garlan, D.: Software Architecture - Perspectives on an emerging discipline. Prentice Hall, 1996.
11. Petriu, D, Woodside, M.: A Metamodel for Generating Performance Models from UML Designs In Proc. of UML 2004 conference, 11-15 October 2004, Lisbon, Portugal. Springer LNCS 3273.

Appendix: Acknowledgement

We extend our thanks to all those who have participated in the organization of this workshop, particularly to the program committee, which is comprised of:

- Arsanjani Ali, IBM Global Services, USA
- Bosch Jan, University of Groningen, the Netherlands
- Dubois Eric, CRP Henri Tudor, Luxembourg
- Egyed Alexander, Teknowledge Corporation, USA
- Ewetz Hans, Clearstream International, Luxembourg
- Garlan David, Carnegie Mellon University, USA
- Issarny Valerie, INRIA, France
- Kruchten Philippe, University of British Columbia, Canada
- Ortega-Arjona Jorge, Universidad Nacional Autonoma de Mexico, Mexico
- Pastor Oscar, Universidad Politecnica de Valencia, Spain
- Poels Geert, University of Arts and Sciences Brussel, Belgium
- Razavi Reza, University of Luxembourg, Luxembourg
- Riehle Dirk, Stanford University, USA
- Romanovsky Alexander, University of Newcastle, UK
- Rosenblum David, University College London, UK
- Sharif Niloufar, Clearstream International, Luxembourg